Долни граници

Тодор Дуков

Какво са долни граници?

Дойде времето за по-депресиращите резултати в курса. До сега единственото, което правихме, беше да показваме, че за задача $\mathbf{X}$ може да се напише алгоритъм със времева сложност $O(f)$ или $\Theta(f)$ за някое $f \in \mathcal{F}$. Доста естествено изниква следния въпрос:

Възможно ли е да съществува по-бърз алгоритъм, който решава задачата $\mathbf{X}$*?*

Ясно е, че е неприемлив отговор от сорта на

Не мога да измисля такъв алгоритъм, следователно такъв алгоритъм не съществува.

В тази тема ще се опитаме да отговорим в някакъв смисъл положително на въпроси от този тип. Преди това нека въведем няколко дефиниции.
Изчислителна задача ще наричаме всяко множество от наредени двойки $\mathbf{X}$, като:

- $\mathrm{Dom}(\mathbf{X})$ ще наричаме **вход**;
- $\mathrm{Rng}(\mathbf{X})$ ще наричаме **изход**.

За пример можем да вземем изчислителната задача за сортиране:

Вход: Целочислен масив $A[1 \dots n]$.
Изход: Пермутация $A'[1 \dots n]$ на $A[1 \dots n]$, за която $A'[1] \leq A'[2] \leq \dots \leq A'[n]$.

Ще казваме, че **алгоритъм AlgX решава задачата X**, ако за всяко $x \in \mathrm{Dom}(\mathbf{X})$ е изпълнено $\langle x, \mathbf{AlgX}(x) \rangle \in \mathbf{X}$.
Нека $\mathbf{X}$ е изчислителна задача и нека $f \in F$. Тогава:

- Казваме, че f е **долна граница** за $\mathbf{X}$, ако всеки алгоритъм, който решава $\mathbf{X}$, работи за време (или брой операции от конкретен вид) поне f[*].
- Казваме, че $\Omega(f)$ е **долна граница** за $\mathbf{X}$, ако всеки алгоритъм, който решава $\mathbf{X}$, работи за време (или брой операции от конкретен вид) $\Omega(f)$.

Техники за изследване на долни граници

Най-често се използват следните техники, които показват че задача $\mathbf{X}$ има долна граница за време f (или $\Omega(f)$):

- директни разсъждения за конкретния пример – тази техника обикновено се използва в малки задачи, където човек за сравнително малко време може да направи пълен анализ. Разбира се, тази техника се използва и при по-обобщените примери, но не толкова често.
- дърво на взимане на решения – тази техника се използва в задачи, където за решаването им се изисква задаването на редица въпроси, чиите отговор ни дава все повече и повече информация. Можем да си мислим за запитванията заедно с информацията, която носят, като едно дърво. Всеки въпрос ще разклонява дървото, докато накрая имаме цялата ни нужна информация в листата, и не трябва да задаваме повече въпроси. Тогава долната граница ще бъде височината на дървото.
- аргументация чрез противник – тази техника е трудна да се обясни без да се даде пример. Идеята е, че играем срещу противник, като нашата цел е да разкрием някаква информация, която уж е била предварително фиксирана. Противника обаче си измисля информацията на момента, като целта му е да ни накара да зададем колкото се може повече въпроси и в отговорите му на въпросите да няма противоречия.
- чрез редукция[†] – ако знаем, че можем алгоритмично да сведем задача $\mathbf{Y}$ до задача $\mathbf{X}$ за време по-малко f и $\mathbf{Y}$ има долна граница за време f (или $\Omega(f)$), то тогава второто е изпълнено и за задача $\mathbf{X}$. В някакъв смисъл тази редукция казва, че задачата $\mathbf{X}$ е поне толкова трудна, колкото задачата $\mathbf{Y}$.

[*] Тук се има предвид, че ако $T(n)$ е броят на стъпки (или операции от конкретен вид), за който алгоритъма завършва при вход с големина n, то $f(n) \leq T(n)$.

[†] Това е може би най-приложимата техника от всички. Тя се използва не само в контекста на сложност. Оказва се, че е много удобно човек да може да говори за това дали една задача е *"по-трудна"* от друга в контекста на разрешимост.

Техниките в действие

Ще започнем със пример за аргументация с противник. Решаваме задачата за максимален елемент – даден ни е като вход целочислен масив $A[1 \dots n]$ и искаме да получим като изход $\max\{A[i] \mid 1 \le i \le n\}$. Твърдим, че всеки алгоритъм, който решава задачата, използва поне $n-1$ сравнения. За целта ще покажем, че във всеки такъв алгоритъм всяко число $A[i]$ участва в някое сравнение. Да допуснем, че има алгоритъм **AlgMax**, за който $A[i]$ не участва в сравнения за някое $1 \le i \le n$.

- Ако **AlgMax** връща $A[i]$, то тогава ако сменим $A[i]$ със $A[i]-1$, **AlgMax** няма да различи по никакъв начин двата входа и ще върне същия резулат, което е абсурд, защото вече трябва да върне нещо друго.

- Ако **AlgMax** връща $A[j]$ за някое $j \neq i$, то тогава ако сменим $A[i]$ със $A[j]+1$, **AlgMax** няма да различи по никакъв начин двата входа и ще върне същия резултат, което е абсурд, защото вече трябва да върне $A[i]$.

Тогава първия елемент ще участва в поне едно сравнение с някой друг, откъдето сме отметнали 2 елемента с 1 сравнение. Останалите също участват в поне едно сравнение (може и с някой от предните два), откъдето за останалите $n-2$ елемента използваме още поне $n-2$ сравнения. Общо стават $n-2+1=n-1$ сравнения. Човек лесно може да напише алгоритъм, който използва точно $n-1$ сравнения.
Нека сега дадем пример за дърво на вземане на решения. Разглеждаме задачата за сортиране. Ще покажем, че всеки сортиращ алгоритъм, който работи на базата на директни сравнения, работи за време $\Omega(n \log(n))$. Нека фиксираме $n \in \mathbb{N}$ и някакъв символ a. Нека $\mathcal{T}_n$ е множеството от всички дървета, за които е изпълнено, че:

- върховете са от вида $(a_i < a_j, P)$, където $1 \le i, j \le n,\ P \subseteq S_n$[‡] и $|P| \ge 2$, или са от вида $\sigma \in S_n$;

- коренът е $(a_i < a_j, P)$ за някои $1 \le i, j \le n$;

- за всеки връх от вида $(a_i < a_j, P)$, има $1 \le k_1, k_2, m_1, m_2 \le n$, за които:
  - лявото му дете е $(a_{k_1} < a_{m_1}, \{\sigma \in P \mid \sigma(i) < \sigma(j)\})$, и
  - дясното му дете е $(a_{k_2} < a_{m_2}, \{\sigma \in P \mid \sigma(i) \not< \sigma(j)\})$;

- всеки връх от вида $\sigma \in S_n$ е листо.

Тогава ако вземем произволен алгоритъм за сортиране **AlgX**, който е базиран на сравнение, при пресмятането на $\mathbf{AlgX}(A[1 \dots n])$, можем да забележим, че траверсираме някое дърво $T \in \mathcal{T}_n$ от корен до листо. В корена се намира първото запитване $a_i < a_j$ (т.е. можем да си мислим, че питаме дали $A[i] < A[j]$), и спрямо отговора на дадено запитване ние се движим наляво или надясно в дървото. Накрая се намираме в листо $\sigma \in S_n$, която задава сортирана пермутация $A'[1 \dots n]$ на $A[1 \dots n]$ така – $A'[i] = A[\sigma(i)]$. Това означава, че за всяко сравнение по време на работа на $\mathbf{AlgX}(A[1 \dots n])$ можем да си мислим, че минаваме през едно ребро в T. В най-лошия случай входа $A[1 \dots n]$ би бил такъв, че да трябва да изминем максимален път от корен до листо т.е. път с дължина $h(T)$. Но T е двоично дърво с $n!$ листа и разклоненост 2, следователно $h(T) \ge \log_2(n!)$. Така в този случай извикването $\mathbf{AlgX}(A[1 \dots n])$ ще използва поне $\log(n!)$ сравнения. Тъй като $\log(n!) \asymp n \log(n)$, получаваме че всеки сортиращ алгоритъм, базиран на сравнения, прави $\Omega(n \log(n))$ сравнения.
Нека сега покажем един пример с редукция. Разглеждаме изчислителната задача **Матрьошки**:
Вход: Масив $T[1 \dots n]$ със елементи от вида (l, w, h) – дължините, широчините, височините на n играчки с форма на правоъгълен паралелепипед, които могат да се вложат една в друга.
Изход: Ред на влагане на играчките, като вътрешната играчка е първа.
Оказва се, че тази задача се решава (на базата на директни сравнения) за време $\Omega(n \log(n))$. Ще покажем това като сведем задачата за сортиране към **Матрьошки**. Нека **AlgM** е алгоритъм, който решава задачата **Матрьошки** със сложност по време $f(n)$. Тогава следният алгоритъм очевидно сортира масива $A[1 \dots n]$:

1. Декларираме нов масив $T[1 \dots n]$.
2. За всяко $1 \le i \le n$ инициализираме $T[i]$ със $(A[i], A[i], A[i])$.
3. Извикваме $\mathbf{AlgM}(T[1 \dots n])$ с резултат $T'[1 \dots n]$.
4. Декларираме нов масив $A'[1 \dots n]$
5. За всяко $1 \le i \le n$ инициализираме $A'[i]$ със $\mathrm{fst}(T'[i])$, където $\mathrm{fst}((a, b, c)) = a$.
6. Връщаме $A'[1 \dots n]$.

Сложността на алгоритъма е $\Theta(n + f(n))$. Ако $f(n) = o(n \log(n))$, щяхме да получим сортиращ алгоритъм, който работи за време $o(n \log(n))$, което е абсурд.

[‡] S_n е симетричната група за $\{1, \dots, n\}$.

Задачи

Задача 1. Един целочислен масив $A[1\dots 2n]$ ще наричаме симетричен, ако за всяко $1 \leq i \leq n$ е изпълнено, че $A[i] = A[2n-i+1]$. Да се докаже, че сортирането на симетрични масиви чрез сравнения изисква време $\Omega(n\log(n))$.

Задача 2. Един целочислен масив $A[1\dots 2n]$ ще наричаме специален, ако за всяко $1 \leq i \leq n$ е изпълнено, че $A[2i] < A[2i-1]$. Да се докаже, че сортирането на специални масиви чрез сравнения изисква време $\Omega(n\log(n))$.

Задача 3. Разглеждаме задачата **PCC**:

Вход: Точки $P_1, \dots, P_n \in \mathbb{N} \times \mathbb{N}$.
Изход: Последователност $P'_1, \dots, P'_n$ на точките $P_1, \dots, P_n$, за която за всяко $1 \leq i < n$ е изпълнено, че най-малкото завъртане от вектора $\overrightarrow{OP'_i}$ към вектора $\overrightarrow{OP'_{i+1}}$ става обратно на часовниковата стрелка, където O е началото на координатната система.

Да се докаже, че задачата **PCC** се решава чрез сравнения за време $\Omega(n\log(n))$.

Задача 4. Нека $A[1\dots n]$ и $B[1\dots n]$ са целочислени масиви. Да се докаже, че $2n-1$ е долна граница за броя на сравнения при сливането на тези два масива в един сортиран масив $C[1\dots 2n]$.

Задача 5. Даден е граф с $2n$ върха. Интересуват ни въпроси от вида:

"Има ли ребро от връх i до връх j?"

Да се докаже, че n^2 е долна граница за броя въпроси, който е нужен, за да установим дали дадения граф е свързан.

Задача 6. Искаме да познаем число между 1 и n, което някой друг човек е намислил. За целта можем да му задаваме въпроси (на които той трябва да отговори честно) от вида:

"Вярно ли е, че k е по-малко от намисленото число?"

Да се докаже, че $\lceil \log_2(n) \rceil$ е долна граница за броя въпроси, който е нужен, да познаем намисленото число.

Задача 7. Да се докаже, че сортирането на двоична пирамида чрез сравнения изисква време $\Omega(n\log(n))$.

Задача 8. Дефинираме вълнист масив индуктивно:

- Всеки едноелементен масив е вълнист.
- Масив $A[1\dots n]$ (където $n > 1$) е вълнист, ако
  1. масивът $A[1\dots \lfloor \frac{n}{2} \rfloor]$ е сортиран, а
  2. масивът $A[\lfloor \frac{n}{2} \rfloor + 1 \dots n]$ е вълнист.

Да се докаже, че пермутирането на масив до вълнист изисква време $\Omega(n\log(n))$.

Задача 9. Да се докаже, че строенето на двоична пирамида от масив $A[1\dots n]$ изисква $n-1$ сравнения.